# Comprenda la diferencia entre var, let y const en JavaScript

Otávio Felipe do Prado
07/10/2020

En la mayoría de los lenguajes de programación, el alcance de las variables locales está vinculado al bloque donde se declaran. Como tal, "mueren" al final de la instrucción en la que se realizan. ¿Esto también se aplica al lenguaje JavaScript? Vamos a revisar:

```
var exibeMensaje = function() {
    var mensajeFueraDelIf = 'Caelum';
    if(true) {
        var mensajeDentroDelIf = 'Alura';
        console.log(mensajeDentroDelIf)// Alura ;
    }
    console.log(mensajeFueraDelIf); // Caelum

    console.log(mensajeDentroDelIf); // Alura
}
```

Estamos declarando dos variables en diferentes bloques de código, ¿cuál será el resultado? Vamos a probar:

```
exibeMensaje(); // Imprime 'Alura', 'Caelum' y 'Alura'
```

Si se declaró `mensajeDentroDelIf` dentro del `if`, ¿por qué todavía tenemos acceso a él fuera del bloque de esta declaración?

## Usar antes de la declaración

A continuación se muestra otro ejemplo de código JavaScript:

```
var exibeMensaje = function () {
    mensaje = 'Alura';
    console.log (mensaje);
    var mensaje;
}
```

Tenga en cuenta que estamos declarando la variable del mensaje solo después de asignar un valor y mostrarlo en el registro, ¿funciona? ¡Vamos a probar!

```
exibeMensaje(); // Imprime 'Alura'
```

¡Funciona! ¿Cómo es posible usar la variable `mensaje` incluso antes de declararla? ¿El alcance está garantizado solo dentro del lugar donde se creó la variable?

## Hoisting

En JavaScript, cada variable es **"elevada"** ***(hoisting)*** a la parte superior de su contexto de ejecución. Este mecanismo mueve las variables a la parte superior de su alcance antes de que se ejecute el código.

En nuestro ejemplo anterior, como la variable `mensaje` está dentro de una *función*, su declaración se eleva (*alzando*) a la parte superior de su contexto, es decir, a la parte superior de la *función*.

Es por esta misma razón que "es posible usar una variable antes de que se haya declarado": en tiempo de ejecución, la variable se elevará (*hoisting*) y todo funcionará correctamente.

## var

Considerando el concepto de *hoisting*, hagamos una pequeña prueba usando una variable declarada con var incluso antes de que se haya declarado:

```
void function() {
    console.log(mensaje);
} ();
var mensaje;
```

En el caso de la palabra clave `var`, además de la variable que se iza(*hoisting*), se inicializa automáticamente con el valor indefinido (si no se asigna ningún otro valor).

De acuerdo, pero ¿cuál es el impacto que tenemos cuando hacemos este tipo de uso?

Imagine que nuestro código contiene muchas líneas y que su complejidad no es tan trivial de entender.

A veces, queremos declarar variables que se usarán solo dentro de una pequeña porción de nuestro código. Tener que lidiar con el alcance de la función de las variables declaradas con `var` (alcance integral) puede confundir las mentes hasta de los programadores más experimentados.

Conociendo las "complicaciones" que pueden causar las variables declaradas con `var`, ¿qué podemos hacer para evitarlas?

## let

Fue pensando en traer el alcance de bloque (tan conocido en otras lenguajes) que ECMAScript 6 tenía la intención de proporcionar la misma flexibilidad (y uniformidad) para el lenguaje.

Usando la palabra clave `let`, podemos declarar variables con alcance de bloque. Vamos a ver:

```
var exibeMensaje = function () {
    if(true)) {
        var scopeFunction = 'Caelum';
        let scopeBlock = 'Alura';
```

```
        console.log (scopeBlock); // Alura
    }
    console.log (scopeFunction); // Caelum
    console.log (scopeBlock);
}
```

¿Cuál será la salida del código anterior?

```
exibeMensaje(); // Imprime 'Alura', 'Caelum' y da un error
```

Tenga en cuenta que cuando intentamos acceder a una variable que ha sido declarada usando la palabra clave `let` dejada fuera de su alcance, se presentó el error *Uncaught ReferenceError: scopeBlock no está definido*.

Por lo tanto, podemos usar `let` sin problemas, ya que el alcance de bloque está garantizado.

## const

Aunque let garantiza el alcance, todavía existe la posibilidad de declarar una variable con `let` y ella ser *undifined*. Por ejemplo:

```
void function () {
  let mensaje;
    console.log (mensaje); // Imprime indefined
} ();
```

Suponiendo que tenemos una variable que queremos garantizar su inicialización con un cierto valor, ¿cómo podemos hacer esto en JavaScript sin causar una inicialización *default* con *undefined*?

Para tener este tipo de comportamiento en una variable en JavaScript, podemos declarar constantes usando la palabra clave `const`. Echemos un vistazo al ejemplo:

```
void function () {
    const mensaje = 'Alura';
    console.log (mensaje); // Alura
    mensaje = 'Caelum';
} ();
```

El código anterior genera un *Uncaught TypeError* : *Assignment to constant variable*, ya que el comportamiento fundamental de una constante es que una vez que se le asigna un valor, no se puede cambiarlo.

Al igual que las variables declaradas con la palabra clave `let`, las constantes también tienen un alcance de bloque.

Además, las constantes deben inicializarse en el momento de la declaración. Aquí hay unos ejemplos:

```
// constante válida
const edad = 18;

// constante inválida: ¿dónde está la inicialización?
const pi;
```

En el código anterior, tenemos el ejemplo de una constante de edad que se declara e inicializa en la misma línea (constante válida) y otro ejemplo en el que el valor no se asigna en la declaración pi (constante no válida) que causa el error *Uncaught SyntaxError*: *Missing initializer in const declaration*.

Es importante usar `const` para declarar nuestras variables, porque de esa manera obtenemos un comportamiento más predecible, ya que el valor que reciben no se puede cambiar.

## Conclusión

Aquí hay un resumen del sitio [constletvar.com](constletvar.com):


Gracias a el *Hoisting*, las variables declaradas con la palabra clave `var` pueden usarse incluso antes de su declaración.

Por otro lado, las variables creadas con `let` solo pueden usarse después de su declaración, porque, a pesar de ser altas, no se inicializan.

Además de las variables declaradas con `var`, tenemos la posibilidad de usar constantes a través de la palabra clave `const` o usar variables con alcance de bloque a través de `let`.

Si te interesa este tema, aquí en Alura tenemos capacitaciones de front-end, para principiantes y para aquellos que ya son desarrolladores web. En ellas, verás cómo programar en Javascript, HTML y CSS para construir sitios web.

ARTÍCULOS DE TECNOLOGÍA > FRONT END

- Acceso a todo el contenido de la plataforma por 6 meses

**¡QUIERO EMPEZAR A ESTUDIAR!**

Paga en moneda local en los siguientes países

**ANUAL**

# US$79,90

un solo pago de US$79,90

- 218 cursos
- Videos y actividades 100% en Español
- Certificado de participación
- Estudia las 24 horas, los 7 días de la semana
- Foro y comunidad exclusiva para resolver tus dudas
- Acceso a todo el contenido de la plataforma por 12 meses